PHP Syntax Core

Variables • Control Flow • Functions • Modules

(plus compact “others” you’ll use daily)

- All examples assume **PHP 8.2+**
- Use `declare(strict_types=1);` for type checking.
- Many PHP syntax elements are reminiscent of **shell script**—just be aware of PHP's more rigid rules, rich features, and object-oriented capabilities which shell scripts don't have.

Variables & Types

```php
<?php declare(strict_types=1);

$course = "ASE 230";  // string
$count  = 3;          // int
$ratio  = 3.14;       // float
$isOk   = true;       // bool
$nil    = null;       // null

// Dynamic reassignment is allowed but avoid mixing types.
$pi = 3.14159;  // float
$pi = "3.14159"; // string (discouraged with strict typing mindset)
?>
```

- PHP variables start with $
- Use typed params/returns/properties (next slides) to keep code safe.

Constants & Enums (Core Extras)

```php
<?php declare(strict_types=1);

const APP_NAME = 'MyApp';
define('MAX_RETRY', 3); // legacy alternative

enum Status: string {
  case Draft = 'draft';
  case Published = 'published';
}
```

- Prefer const in code; define() exists for historical reasons.
- Enums (PHP 8.1+) replace ad-hoc string constants for state.

```php
// Create enum instance
$postStatus = Status::Draft;

// Compare
if ($postStatus === Status::Draft) {
  echo "Post is in draft mode\n";
}

// Switch / match
$message = match($postStatus) {
  Status::Draft => "Work in progress",
  Status::Published => "Publicly visible",
};

echo $message;   // "Work in progress"

// Access backing value
echo $postStatus->value;   // "draft"
```

Strings & Interpolation

```php
<?php declare(strict_types=1);

$name = "PHP";
echo "Hello $name\n";         // interpolation
echo 'Hello $name';           // literal; no interpolation

$heredoc = <<<TXT
Multi-line with $name
TXT;

$nowdoc = <<<'TXT'
Multi-line, no interpolation: $name
TXT;
```

Arrays (Indexed & Associative)

```php
<?php declare(strict_types=1);

$colors = ['red', 'green', 'blue'];                    // indexed
$user   = ['id' => 7, 'name' => 'Ada'];                // associative

$colors[] = 'purple';                                   // push
[$first, $second] = $colors;                            // destructuring
$merged = [...$colors, 'black'];                        // spread (7.4+)
?>
```

- Arrays unify lists & maps
- Use objects/DTO (Data Transfer Object)s when structure matters.

Control Flow — if, elseif, else, Ternary, Null-Coalesce

```php
<?php declare(strict_types=1);

$score = 87;

if ($score >= 90) {
  $grade = 'A';
} elseif ($score >= 80) {
  $grade = 'B';
} else {
  $grade = 'C';
}

$label = ($score >= 60) ? 'pass' : 'fail'; // ternary
$user  = $_GET['u'] ?? 'guest';            // null coalescing
?>
```

Control Flow — match (Expression)

```php
<?php declare(strict_types=1);

$ext = 'png';

$mime = match ($ext) {
  'png'   => 'image/png',
  'jpg', 'jpeg' => 'image/jpeg',
  'gif'   => 'image/gif',
  default => 'application/octet-stream',
};
?>
```

- match is exhaustive and returns a value (no fall-through, safer than switch).

Loops — while, for, foreach, break/continue

```php
<?php declare(strict_types=1);

$i = 0;
while ($i < 3) { $i++; }

for ($j = 0; $j < 3; $j++) { /* ... */ }

$nums = [10, 20, 30];
foreach ($nums as $idx => $n) {
  if ($n === 20) continue;
  if ($idx === 2) break;
}
?>
```

Functions — Signatures & Returns

- PHP function gets arguments, processes the arguments, and returns.

```php
<?php declare(strict_types=1);

function add(int $a, int $b): int {
  return $a + $b;
}
```

Functions — By-Reference, Variadics, Arrow & Closures

- The argument $a is not changed.
- We call this "pass by value".

```php
<?php declare(strict_types=1);

// --- Pass by value (default)
function incVal(int $x): void {
  $x++;
}
$a = 1;
incVal($a);
echo $a; // 1 (unchanged)
```

- When we need to change the argument, we prepend &.
- We call this "pass by reference".

```
// --- Pass by reference
function incRef(int &$x): void {
  $x++;
}
$b = 1;
incRef($b);
echo $b; // 2 (changed)
```

- Use `...` when the number of arguments is unknown (variadic arguments).
- Inside the function, they are collected into an **array**.
- You can iterate, map, or reduce as needed.

```php
// --- Variadic arguments
function sumAll(int ...$nums): int {
  return array_sum($nums);
}
echo sumAll(1, 2, 3, 4); // 10
?>
```

- We can define functions using closure and arrow.
- We make a function using `function` keyword and body (closure functions).
- Closure functions are multi-line, explicit return.

```php
<?php>
$triple = function(int $n): int {
  return $n * 3;
};
echo $triple(5); // 15
?>
```

- When we need simple one-line function, we use `fn` and `=>` (arrow functions).
- Arrow functions are one-line, implicit return

```php
<?php declare(strict_types=1);

// Arrow fn (short syntax, implicit return)
$double = fn(int $n): int => $n * 2;
echo $double(5); // 10
?>
```